Computer Architecture

Project 3

Due Date : 2022, 05, 26 23:55 PM

Project 3) Cache Simulation with gem5

다음 지시사항을 지켜 주시길 바랍니다.

1) project3_1, project3_2폴더를 만든 뒤, 여기에 각각 필요한 파일들을 저장해 주세요. 해당 내용은 3-1, 3-2 설명에 포함돼 있습니다.
2) 두 폴더를 같이 tar파일로 압축해 주세요. 압축 파일명은 반드시 project3_학번.tar 가 돼야 합니다.
   예) project3_2015147526.tar

※ 프로젝트 구현을 위한 소스 코드들을 함께 첨부하였습니다.
configs/common/CacheConfig.py (덮어씌울것)
configs/common/Options.py (덮어씌울것)

첨부된 두 파일을 Project2에서 사용한 gem5의
configs/common/ 폴더에 복사해주세요.

src/mem/cache/tags/my_policy.cc
src/mem/cache/tags/my_policy.hh
src/mem/cache/tags/SConscript (덮어씌울것)
src/mem/cache/tags/Tags.py (덮어씌울것)

첨부된 네 파일을 Project2에서 사용한 gem5의
src/mem/cache/tags/ 폴더에 복사해주세요.

본 과제에선 my_policy.cc 파일을 수정하셔야 합니다.
my_policy.cc의 findVictim함수에서
WRITE_YOUR_CODE_IN_HERE
이렇게 주석처리된 부분을 여러분들께서 작성해주시면 됩니다.
(추가로 변수가 필요할 시 my_policy.hh에서 주석처리된 부분을 확인해주세요).

※작성이 완료된 후, 반드시 scons build/ARM/gem5.opt (project 2에서 다른 커맨드로 빌드하신 분은 그 때 사용하셨던 커맨드)를 다시 돌리셔야 내용이 반영이 됩니다.
※또한, 해당 부분에서 blk값을 수정하기 전까지는 제대로 위의 scons 커맨드를 쓰셔도 빌드하실 수 없습니다. (blk값이 설정이 되지 않은 채로 사용되므로 에러가 납니다. blk를 사용하도록 my_policy.cc를 수정하셔야 에러가 나지 않습니다.)

Project 3-1

- gem5를 실행할 때, se.py에 --l1d_assoc=n(숫자) 옵션을 추가하여 n-way associative cache를 설정할 수 있습니다.
- 여기에 더불어, gem5에선 기본적으로 LRU, Random replacement 등의 cache replacement policy가 구현되어 있습니다.
- 이번 과제에선, --dcache-repl-policy 옵션을 통해 데이터 캐쉬의 교체 정책을 옵션으로 수정할 수 있습니다 (RandomRepl, LRU, MY_POLICY(직접 구현))
- 이와 더불어, MY_POLICY replacement policy를 직접 구현해주세요. MY_POLICY는 Not Most Recently Used (NMRU) policy를 따릅니다.
  (첨부된 src/mem/cache/tags/my_policy.cc 및 src/mem/cache/tags/my_policy.hh)
  MY_POLICY(NMRU)의 교체용 블록을 찾는 조건(findVictim 함수)
  1) Set 내에 Invalid 한 블록이 있다면 해당 블록을 교체
  2) Set 내의 모든 블록이 Valid하다면, 가장 최근에 사용된 블록(MRU, Most Recently Used)을 제외한 블록중에 랜덤한 블록을 교체용 블록으로 삼음
- 기본 캐쉬 크기는 64kB로 실행할 프로그램에 비해 너무 큽니다. 이번 과제에선 Associativiy와 교환 정책에 따른 영향을 보기 위해, 캐쉬 사이즈를 줄여서 시뮬레이션을 진행하겠습니다. --l1d_size 옵션을 사용하시면 됩니다. **--l1d_size=1kB** (kB입니다)

Config #	L1 Data Cache	
	Associativity	Replacement Policy
0	1	LRU (Policy 영향 X)
1	2	LRU (Default)
2	4	LRU (Default)
3	8	LRU (Default)
4	2	Random
5	4	Random
6	8	Random
7	2	MY_POLICY (NMRU)
8	4	MY_POLICY (NMRU)
9	8	MY_POLICY (NMRU)

위 표에 맞춰, 10번에 걸쳐 mibench의 bitcount 벤치마크와 sha 벤치마크를 실행한 뒤, 각 실행의 config.ini 파일과 stat.txt 파일을 제출하세요. 실행 환경은 다음과 같습니다.

- mibench 빌드방법은 Project2를 참고해 주세요.
- --cpu-type은 arm_detailed를 사용해 주세요.
- cpu-type 뒤에 --caches 옵션을 추가하여 캐쉬를 사용해 주세요.
- --l1d_size=1kB 옵션을 통해 캐쉬 사이즈를 1kB로 설정해 주세요.
- bitcount는 -o 뒤의 입력값을 Project 2와 동일하게 75000이 아닌 3000을 사용해 주세요.
- sha 역시 Project 2와 동일하게 sha_50kbInput.asc 파일을 인풋으로 돌려 주시길 바랍니다.
- bitcount와 sha를 반드시 -O3 옵션으로 빌드해주시길 바랍니다. (기본 Makefile에 포함되어 있습니다)
- 구현 Hint) 아래의 두 Policy를 참고하세요
  Random policy(src/mem/cache/tags/random_repl.cc)는
  1) Set내에 Invalid한 블록이 있는지 검사
  2) Set내에 Invalid한 블록이 있다면 이를 교체
  3) Set내에 Invalid한 블록이 없다면 랜덤한 블록을 교체
  위와 같은 방법으로 Victim을 고릅니다.

  LRU policy(src/mem/cache/tags/lru.cc)는
  - Access 혹은 Insert된 블록을 해당 set 리스트의 맨 앞으로 이동
  - Invalidate된 블록을 해당 set 리스트의 맨 뒤로 이동
  - Victim을 고를 때 set 리스트의 맨 뒤를 선택
  위와 같은 구현으로 구현되어 있습니다. 또한 my_policy.cc는 LRU와 완전히 동일한 Access/Insert/Invalidate 함수를 사용합니다. random_replc.cc 및 lru.cc의 코드를 분석해 보시길 바랍니다.

Project 3-1 제출 파일

- project3_1 폴더 안에 bitcount 폴더와 sha 폴더를 만들어 주세요.
- 각각의 폴더에, 실험 숫자에 맞춰 (0~9, 위 표 참조) config_n.ini 파일 및 stats_n.txt 파일을 제출해 주세요.
- TIP) config.ini파일은 -d 옵션으로 지정한 폴더에 자동으로 저장되므로, 같은 폴더에서 여러 번 실행하면 파일이 덮어 씌워질 수 있으니 주의하시길 바랍니다. 아예 각 실행을 다른 폴더에서 하신 뒤 제출을 위해 각 config/stats파일을 복사하시는 것을 추천드립니다.
- project3_1 폴더 안에 src폴더를 만들고, 여기에 my_policy.cc 및 my_policy.hh 파일을 넣어 제출해 주세요.

Project 3-1) 구현이 정확히 됐는지 확인하기 위해 디버그 하기

● 실행 커맨드라인 예시

※ 반드시 my_policy.cc를 작성 완료하신 후, scons build/ARM/gem5.opt (project 2에서 문제가 발생하셨던 분은 그 때 사용하셨던 커맨드)를 돌리신 후에 실행하셔야 합니다. my_policy.cc를 수정하실 때마다 위의 scons 커맨드를 다시 돌려주셔야 반영됩니다.

Config 1 (LRU, assoc=2 예시)

```
./build/ARM/gem5.opt -d bitcount/config_1 -re
--stdout-file=simout_1 --stderr-file=simerr_1 --stats-file=stats_1
./configs/example/se.py --cpu-type=arm_detailed  -c
/home/sohwisoo/miBench/automotive/bitcount/bitcnts_ARM_O3
-o 10000 --output=bitcount/config_1/result_1 --caches
--l1d_size=1kB --dcache-repl-policy=LRU --l1d_assoc=2
```

Config 5 (Random, assoc=4 예시)

```
./build/ARM/gem5.opt -d bitcount/config_5 -re
--stdout-file=simout_5 --stderr-file=simerr_5 --stats-file=stats_5
./configs/example/se.py --cpu-type=arm_detailed  -c
/home/sohwisoo/miBench/automotive/bitcount/bitcnts_ARM_O3
-o 10000 --output=bitcount/config_5/result_5 --caches
--l1d_size=1kB --dcache-repl-policy=Random --l1d_assoc=4
```

Config 9 (MY_POLICY, assoc=8) 예시

```
./build/ARM/gem5.opt -d bitcount/config_9 -re
--stdout-file=simout_9 --stderr-file=simerr_9 --stats-file=stats_9
./configs/example/se.py --cpu-type=arm_detailed  -c
/home/sohwisoo/miBench/automotive/bitcount/bitcnts_ARM_O3
-o 10000 --output=bitcount/config_9/result_9 --caches
--l1d_size=1kB --dcache-repl-policy=MY_POLICY --l1d_assoc=8
```

● 실행이 설정한 옵션으로 됐는지 확인하는 방법

--> config.ini의 [system.cpu.dcache.tags] 부분을 확인해 보시면

type (여기에 LRU, Random, MY_POLICY 등의 정책이 들어갑니다)

assoc (associativity)

size (캐시사이즈를 나타내며, 이번 과제에선 1024여야 합니다)

등을 확인하실 수 있습니다.

● MY_POLICY가 제대로 구현됐는지 확인하기
※ 반드시 my_policy.cc를 작성 완료하신 후, scons build/ARM/gem5.opt (project 2에서 문제가 발생하셨던 분은 그 때 사용하셨던 커맨드)를 돌리신 후에 실행하셔야 합니다. my_policy.cc를 수정하실 때마다 위의 scons 커맨드를 다시 돌려주세요.
이번 과제에서는 디버그를 위해 CA_DEBUG라는 디버그 플래그를 추가하였습니다. MY_POLICY로 실행하실 때, 커맨드라인에 이를 추가해주시면 set index가 0인 캐시의 insert/access/invalidate를 확인하실 수 있습니다.

커맨드라인 예시
./build/ARM/gem5.opt -d bitcount/config_9_debug -re
--debug-file=debug_9 --debug-flags=CA_DEBUG
--stdout-file=simout_9 --stderr-file=simerr_9 --stats-file=stats_9
./configs/example/se.py --cpu-type=arm_detailed -c
/home/sohwisoo/miBench/automotive/bitcount/bitcnts_ARM_O3
-o 10000 --output=bitcount/config_9_debug/result_9 --caches
--l1d_size=1kB --dcache-repl-policy=MY_POLICY --l1d_assoc=8

debug 파일을 바탕으로 (위의 커맨드라인 예시에선 debug_9),
여러분들이 구현하신 MY_POLICY가 정말 스펙대로 동작하는지 확인해보시길 바랍니다.

Project 3-2
3-1의 결과를 바탕으로, 두 벤치마크별로 각 설정에 따른 miss rate의 그래프를 그리고, 어느 설정이 가장 성능/데이터캐시의 miss rate 관점에서 뛰어난 지를 간단하게 분석해주세요. stats 파일의
system.cpu.dcache.overall_miss_rate::total
system.cpu.dcache.overall_misses::total
system.cpu.dcache.overall_hits::total
값 등을 확인하시길 바랍니다. 프로그램의 실제 성능은
final_tick
값을 바탕으로 비교하시면 됩니다.

※기본 스펙에선 miss rate 관점에서만 분석하시면 됩니다. 혹시 추가로 다른 부분들을 살펴보시고 싶으시다면 grep 및 vimdiff를 이용해 보시길 바랍니다.

Project 3-2 제출 파일
project3_2 폴더 안에 report.pdf 파일로 그래프 및 분석을 기술해 주세요.